Web Services and Cloud-Based Systems Assignment 1.2

REST calculator service report. Martin Warnaar (10689613) and Bryan te Beek (10690441)

The REST calculator service we made for this assignment is written in ruby using Sinatra (http://www.sinatrarb.com/). This ruby framework allows quickly creating web applications by specifying handlers for HTTP requests. The application consists of 2 parts: calc and calc2. Calc allows clients to make requests with an equation and returns the result of the equation, or an error message if it could not compute the outcome. Calc 2 is a stateful implementation of this, allowing the client to store outcomes of computations on the server and using the stored outcomes to make further calculations with.

Calc

Calc handles just one type of request, namely GET requests with equations that are in RPN (Reverse Polish Notation). We didn't want to figure out how to actually evaluate the RPN expressions, so we found a nice class that we could use: http://stackoverflow.com/questions/17417175/refactoring-feedback-for-reverse-polish-notation-rpn-or-postfix-notation. Now we just need to create an instance of the class and call the evaluate function to get back the result like this:

calc = RPNCalculator.new
value = calc.evaluate(equation)

Next we check if the result equals infinity, and return "division by zero" with a status code of 400 to the client if it is. If it doesn't throw some exception and the value is not infinity, it returns the result to the client with the default status code (200). Finally, if something else went wrong it must have been a syntax error. We catch this in the rescue clause and return "syntax error" also with status 400.

Calc2

For the stateful implementation of the calculator service, we used the same RPNCalculator class to evaluate RPN expressions. We keep track of calculation variables in a ruby Hash (http://www.ruby-doc.org/core-2.1.1/Hash.html) containing key-value pairs "ID => value". To insert a new calculation variable, the client performs a POST request to the server with an equation. This equation is evaluated and the outcome is stored using a random generated ID with (0...8).map { (65 + rand(26)).chr }.join if it is a syntactically correct expression. The server returns this ID or the error message with the according status code to the client. When a client performs a GET request with this ID like '/calc2/:id', the value currently stored for that ID is returned by using the specified ID as key like before, or a 404 status code when it is not contained in the Hash. When a client performs a GET request to '/calc2/' without an ID, the list of keys (IDs) is returned, separated by commas using array.join(","), array being the array of keys

in the Hash. Delete can be called by the client performing a DELETE request to ‘calc2/:id’. In the same way as before, the ID is used as key to call delete on the Hash as follows:
deletedvalue = keyvalues.delete(id)
Hash.delete() returns the deleted value, so we can use the deletedvalue variable to check whether something actually was deleted or not, and return the according status code. DELETE requests to ‘calc2’ without an ID removes everything currently in the Hash and returns a 204 status.
Finally, when a client performs a PUT request with an ID and equation containing “ACC”, the RPN expression is evaluated by replacing “ACC” with the value currently stored for that ID and storing it in the Hash. Then the result of the equation with the stored variable is returned to the client, or an error message as described before.

Testing

Both calc and calc2 were tested succesfully using the test applications on https://github.com/mikolajb/soa-cloud-course.

Calc

```
ruby client.rb --url http://localhost:4567/calc/ --notation rpn
```

Calc2

```
ruby client-state.rb --url http://localhost:4567/calc2/ --notation rpn
```